UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/753,766	12/29/2000	Udo Walterscheidt	2207/10124	3472

7590 04/20/2005

KENYON & KENYON
Suite 600
333 W. San Carlos Street
San Jose, CA 95110-2711

EXAMINER	
LI, AIMEE J	
ART UNIT	**PAPER NUMBER**
2183	

DATE MAILED: 04/20/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
UNITED STATES PATENT AND TRADEMARK OFFICE
P.O. BOX 1450
ALEXANDRIA, VA 22313-1450
www.uspto.gov

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application Number: 09/753,766
Filing Date: December 29, 2000
Appellant(s): WALTERSCHEIDT ET AL.

Paul E. Steiner
<u>For Appellant</u>

EXAMINER'S ANSWER

This is in response to the appeal brief filed 14 January 2005.

(1) Real Party in Interest

11

A statement identifying the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

A statement identifying the related appeals and interferences which will directly affect or be directly affected by or have a bearing on the decision in the pending appeal is contained in the brief.

(3) Status of Claims

The statement of the status of the claims contained in the brief is correct.

(4) Status of Amendments

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

(5) Summary of claimed subject matter

The summary of invention contained in the brief is correct.

(6) Grounds of rejection to be reviewed on appeal

The appellant's statement of the grounds of rejection in the brief is correct.

(7) Claims Appealed

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Prior Art of Record

5,933,627	Parady	8-1999
5,881,277	Bondi et al.	3-1999
6,567,839	Borkenhagen et al.	5-2003

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

Claims 1-4, 6-9 and 14-15 are rejected under 35 U.S.C. 103(a) as being unpatentable over Parady, U.S. Patent No. 5,933,627, in further view of Bondi et al., U.S. Patent No. 5,881,277.

Regarding claims 1, 8 and 14, taking claim 1 as exemplary, Parady has taught a multi-threading processor, comprising:

> A front-end module (14 of Fig.3),
>
> An execution module (41 of Fig.3) coupled to said front end module,
>
> A state module coupled to said front-end module and said execution module (see 48, 50, 110 of Fig.3). While not shown explicitly as a module, the state module's function, as described in the Applicant's specification, is encompassed by 48, 50, and 110 of Fig.3, which further shows the required interconnections,
>
> A switch logic module (112 of Fig.3) coupled to said state module, wherein said switch logic module detects a long-latency event in a software thread and schedules a switch to another software thread during a latency of said long-latency event (see Col.3 lines 57-65).

Parady has not specifically taught that the switch logic detects a mispredicted branch in a thread and schedules a switch to another thread during the latency of the mispredicted branch. However, Parady has taught that the switch logic module detects a long-latency event in a

software thread and schedules a switch to another software thread during a latency of said long-latency event (Parady Col.3 lines 57-65). Bondi has taught that a branch misprediction event will result in large performance penalties, costing many cycles in order to fetch the correct path's instructions (Bondi Col.1 lines 47-64). One of ordinary skill in the art would have recognized that increasing processor performance and throughput are of paramount concern to designers and, as taught in Bondi, a branch misprediction is a serious impediment to increasing processor performance that needs to be addressed (Bondi column 1, lines 61-67). Therefore, one of ordinary skill in the art would have found it obvious to modify Parady to detect mispredicted branch events and schedule a switch to another software thread, in a manner similar to how Parady handles other long-latency events, in order to reduce the performance penalty suffered by the misprediction.

Claims 8 and 14 are nearly identical to claim 1. They differ in their lack of explicit hardware modules, but are both methods that encompass the same scope as claim 1. Therefore, claims 8 and 14 are rejected for the same reasons as claim 1.

Regarding claim 2, Parady in view of Bondi has taught a multi-threading processor as recited in claim 1, wherein the switch logic module detects a switching event (see Parady, Col.3 lines 57-65).

Regarding claim 3, Parady in view of Bondi has taught a multi-threading processor as recited in claim 2, wherein the switch logic module includes a cache miss indicator that is set when a cache miss is detected and reset when the switch is completed (see Parady, 114 of Fig.3 and Col.3 lines 57-64), but has not explicitly taught a mispredicted indicator that is set when a branch is mispredicted.

However, Bondi has taught a mispredicted signal (see Fig.2, and Col.6 lines 17-25) which is asserted when a branch is determined to have been mispredicted, and deasserted when the prediction is correct. Parady in view of Bondi, as shown above, has taught the detection of mispredicted branch events in order to minimize the performance penalty suffered by a misprediction. One of ordinary skill in the art would have recognized that the only way to detect a mispredicted branch is to execute the conditional branch instruction to get the result and compare it against the prediction. Therefore, one of ordinary skill in the art would have found it obvious to modify Parady to include the misprediction signal of Bondi in order to correctly detect a mispredicted branch.

Regarding claim 4, Parady in view of Bondi has taught a multi-threading processor as recited in claim 3, wherein the switch logic module includes an outstanding switch request indicator that is set when the switching event does not require an immediate switch (see Parady, Col.1 lines 45-57 and Col.4 lines 53-62). For the case of a non-blocking load, there does not need to be an immediate switch. While not taught explicitly, there inherently must be a signal that is set to distinguish between blocking and non-blocking loads so that the processor knows if it can continue executing. This can be considered an outstanding switch request indicator because if it is a blocking load, this signal will be deasserted and not affect the normal thread switching operations.

Regarding claim 6, Parady in view of Bondi has taught a multi-threading processor as recited in claim 1, wherein the state module includes a pair of register files (48 and 50 of Fig.3) and a pair of IPs (110 of Fig.3).

Regarding claim 7, Parady in view of Bondi has taught a multi-threading processor as recited in claim 6, wherein the IPs are coupled to the front-end module and the register files are coupled to the execution module (see Fig.3).

Regarding claims 9 and 15, taking claim 9 as exemplary, Parady in view of Bondi has taught a method for concealing switch latency in a multi-threading processor as recited in claim 8, further comprising executing a switch to another software thread if the switching event requires an immediate switch (see Parady, Col.1 lines 45-57 and Col.4 lines 53-62).

Claim 15 is nearly identical to claim 9. It differs only in its parent claim, but encompasses the same scope. Therefore, claim 15 is rejected for the same reasons as claim 9.

Claims 5, 10-13 and 16-19 are rejected under 35 U.S.C. 103(a) as being unpatentable over Parady, U.S. Patent No. 5,933,627, in further view of Bondi et al., U.S. Patent No. 5,881,277 as applied to claims 1-3 above, and further in view of Borkenhagen et al., U.S. Patent No. 6,567,839.

Regarding claims 5, 10 and 16, taking claim 5 as exemplary, Parady in view of Bondi has taught a multi-threading processor as recited in claim 4, but has not taught wherein the switch logic module includes a counter to schedule a switch based on a time quantum.

However, Borkenhagen has taught the scheduled switching between threads based on a time quantum and a countdown register to prevent threads from being inactive too long while processing a long-latency event (see Col.5 lines 50-57, Col.6 lines 32-35, and Col.18 lines 15-21). One of ordinary skill in the art would have recognized that inactive or stalled threads are wasting valuable processor cycles that could be spent processing other data. Therefore, one of ordinary skill in the art would have found it obvious to modify the multi-threaded processor as

taught by Parady in view of Bondi to include the scheduled thread switching based on the a time quantum of Borkenhagen in order to force inactive threads to switch to new active threads, thus improving processor throughput and reducing wasted processor cycles.

Claims 10 and 16 are nearly identical to claim 5. While they differ in their parent claims, both claims encompass the scope of claim 5. Therefore, claims 10 and 16 are rejected for the same reasons as claim 5.

Regarding claims 11 and 17, taking claim 11 as exemplary, Parady in view of Bondi, in further view of Borkenhagen has taught a method for concealing switch latency in a multi-threading processor as recited in claim 10, wherein the switch takes place "rapidly" (see Borkenhagen, Col.4 lines 38-41). Parady in view of Bondi in further view of Borkenhagen has not explicitly taught wherein the switch has a latency of about 15 to about 20 clocks.

However, at the time the invention was made, it would have been obvious to a person of ordinary skill in the art to make a switch with a latency of about 15 to about 20 clocks because the Applicant has not disclosed that doing so provides an advantage, is used for a particular purpose, or solves a stated problem. Furthermore, one of ordinary skill in the art would have expected Applicant's invention to perform equally well with either the "rapidly" switching between threads taught by Borkenhagen or the claimed about 15 to about 20 clocks because both latencies perform the same function of providing a thread switch with minimal latency. Therefore, one of ordinary skill in the art would have found it obvious to modify the processor of Parady in view of Bondi in further view of Borkenhagen to obtain the invention as specified in claim 11.

Claim 17 is nearly identical to claim 11. It differs only in its parent claim, but encompasses the same scope. Therefore, claim 17 is rejected for the same reasons as claim 11.

Regarding claims 12 and 18, taking claim 18 as exemplary, Parady in view of Bondi, in further view of Borkenhagen has taught a method for concealing switch latency in a multi-threading processor as recited in claim 11, wherein the time quantum can be customized according to a specific hardware configuration (see Borkenhagen, Col.18 lines 15-21, 36-38). Parady in view of Bondi in further view of Borkenhagen has not explicitly taught wherein the time quantum is less than about 1,000 clocks.

However, at the time the invention was made, it would have been obvious to a person of ordinary skill in the art to make the time quantum less than about 1,000 clocks because the Applicant has not disclosed that doing so provides an advantage, is used for a particular purpose, or solves a stated problem. Furthermore, one of ordinary skill in the art would have expected Applicant's invention to perform equally well with either a "customized" time quantum or the claimed less than about 1,000 clocks because both quantums perform the same function of providing a time-out value to force a thread switch. Therefore, one of ordinary skill in the art would have found it obvious to modify the processor of Parady in view of Bondi in further view of Borkenhagen to obtain the invention as specified in claim 12.

Claim 18 is nearly identical to claim 12. It differs only in its parent claim, but encompasses the same scope. Therefore, claim 18 is rejected for the same reasons as claim 12.

Regarding claims 13 and 19, taking claim 19 as exemplary, Parady in view of Bondi in further view of Borkenhagen has taught a method for concealing switch latency in a multi-threading processor as recited in claim 12, wherein the time quantum can be customized

according to a specific hardware configuration (see Borkenhagen, Col.18 lines 15-21, 36-38). Parady in view of Bondi in further view of Borkenhagen has not explicitly taught wherein the time quantum is about 200 clocks.

However, at the time the invention was made, it would have been obvious to a person of ordinary skill in the art to make the time quantum is about 200 clocks because the Applicant has not disclosed that doing so provides an advantage, is used for a particular purpose, or solves a stated problem. Furthermore, one of ordinary skill in the art would have expected Applicant's invention to perform equally well with either a "customized" time quantum or the claimed about 200 clocks because both quantums perform the same function of providing a time-out value to force a thread switch. Therefore, one of ordinary skill in the art would have found it obvious to modify the processor of Parady in view of Bondi in further view of Borkenhagen to obtain the invention as specified in claim 13.

Claim 19 is nearly identical to claim 13. It differs only in its parent claim, but encompasses the same scope. Therefore, claim 19 is rejected for the same reasons as claim 13.

In order to clarify how Applicant's claims correspond to the prior art, the examiner has mapped the limitations of the claims argued in part (vii) Arguments of the Appeal Brief to the device described in the prior art with further explanation of the cited art where deemed appropriate.

In regards to claim 1:

Claim Limitation	Cited Prior Art and Explanations
A multi-threading processor, comprising:	**Parady**: *Column 2, lines 18-20* "The present invention provides a method and apparatus for switching between threads of a program in response to a long-latency event." *Column 3, lines 36-37* "FIG. 3 illustrates portions of the

	processor of FIG. 1"
A front end module	The front end module, as stated in the specification on page 4, lines 18-19, is "...front end module 18 receives and decodes an input containing instructions while performing the necessary diagnostics." **Parady**: *Column 3, lines 11-18* "Instructions from decode unit 14..." *Column 3, lines 37-43* "...a decode unit 14 is the same as in FIG. 1..." *Figure 3, element 14* The decode unit of Parady decodes the instruction and send it for storage in the correct thread buffer, thereby decoding a received instruction and, in order to know which buffer to use, performing the necessary diagnostics.
An execution module	**Parady**: *Column 3, lines 19-25* "The first three functional units..." *Figure 3, element 41* "Execution units"
Said execution module coupled to said front end module	Applicant's claim language states that there is a connection between the execution module and front end module, but does not state whether the connection is a direct connection, i.e. there are no elements between the execution module and front end module so instructions travel directly from the front end module to the execution units, or a indirect connection, i.e. there can be elements between the execution module and front end module so instructions can travel through other elements before reaching the execution module. **Parady**: *Figure 3, elements 14 and 41* Parady shows that the execution units receive the instructions decoded by the decode unit via an instruction buffer, dispatch unit, and the busses/connections between these elements.
A state module	The state module, as stated in the specification on page 4, lines 10-11, is "...a state module 12 having a pair of instruction pointers (IP) 14a and 14b and a pair for register files 16a and 16b." There is no indication in the specification or claims that the state module must be one element and cannot be split over multiple elements. In fact, the state module is made up of multiple elements according to the cited passage in the specification above. **Parady**: *Column 2, lines 25-34* "...separate groups of registers

	for multiple threads, a group of program address registers pointing to different threads…" *Column 3, lines 44-56* "Integer register file 48 is divided up into four register files to support threads 0-3. Similarly, floating point register file 50 is broken into four register files to support threads 0-3…The present invention adds four program address registers 110 for threads 0-3. The particular thread address pointed to will provide the starting address for the fetching instructions to the appropriate one of the instruction buffers…" *Figure 3, elements 38, 50, and 110* Parady shows PA registers 110, Integer Register Files 48, and FP Register Files 50. The PA registers 110 and the instruction pointers are the same element, since they are referring to and storing the address of the currently executing instruction in each thread. The Integer Register Files 48 and FP Register Files 50 are the pair of register files. Therefore, the PA Registers 110, Integer Register Files 48, and FP Register Files 50 make the state module.
Said state module coupled to said front end module and said execution module	Applicant's claim language states that the state module is coupled to the front end and execution modules, but does not state whether the connection is a direct connection, i.e. there are no elements between the state module and the front end and execution modules so information from the modules flow directly from one module to another module, or an indirect connection, i.e. there can be elements between the state module and the front end and execution module so information from one module may travel through other elements before reaching the other module. **Parady**: *Figure 3, elements 48, 50, 110, 41, and 14* Parady shows that there is a direct connection between the Integer Register Files 48 and FP Register Files 50 to Execution Units 41. The PA Registers 110 connect to decode unit 14 via the instruction buffers 102, 104, 106, and 108. Therefore, at least one element of the state module in Parady connects to the front end or execution module, so the state module is coupled to the front end and execution modules.
A switch logic module	**Parady**: *Column 2, lines 27-29* "A switching mechanism switches between the program address registers in response to the long-latency events." *Column 3, lines 57-65* "Thread switching logic 112…"

	Figure 3, element 112
Said switch logic module coupled to said state module	Applicant's claim language states that the switch logic module is coupled to the state module, but does not state whether the connection is a direct connection, i.e. there are no elements between the switch logic module and state module so information flows directly from one module to another module, or an indirect connection, i.e. there can be elements between the switch logic module and the state module so information from one module may travel to other elements before reaching the other module. **Parady**: *Figure 3, elements 112, 110, 48, and 50* Parady shows a direct connection from the thread switching logic 112 to the PA registers 110, Integer Register Files 48, and FP Register Files 50.
Wherein said switch logic module detects a mispredicted branch in a software thread and schedules a switch to another software thread during a latency of said mispredicted branch	Applicant's claim language states that the switch logic: 1) detects a mispredicted branch in a software thread and 2) schedules a switch to another software thread during a latency of said mispredicted branch Detecting a mispredicted branch, according to the broadest reasonable interpretation, is determining whether there is whether there is a mispredicted branch. Scheduling a switch, according to the broadest reasonable interpretation, is appointing the switch to execute at a certain time. According to the Applicant's claim, the certain time is during a latency of said mispredicted branch. So, the switch is to occur when a mispredicted branch stalls, i.e. delays, the pipeline. Also, the Applicants themselves distinguish two types of switch events: 1) events that must be switched immediately 2) events that can be rescheduled to a later point in time Applicant states on page 5, lines 24-26 that "a switching event that must be switched immediately is typically an event that requires a long latency memory access and would otherwise stall the processor and result in wasted idle cycles." Applicant further gives an example of an event that can be rescheduled and an event that must be switched immediately on page 6, line 31 to page 7, line 6. To summarize the example, a load instruction switch event can be rescheduled when it is a "non-blocking" load, but, when the data resulting from the load is needed, an immediate switch must occur. This is because the processor must stall, i.e. stop, execution until the load requiring an immediate switch is resolved. This produces a number of

	wasted cycles that the processor is idle. A mispredicted branch is similar to a load requiring an immediate switch, because the correct next instructions to be executed are unknown and numerous cycles are needed to invalidate the currently erroneously executing instructions and determine the correct instructions to execute in order to resolve the branch misprediction. **Parady**: *Column 2, lines 18-19* "The present invention provides a method and apparatus for switching between threads of a program in response to a long-latency event." *Column 2, lines 27-29* "A switching mechanism switches between the program address registers in response to the long-latency events." *Column 3, lines 58-63* "The indication that a thread switch is required is provided on a line 114...Upon such an indication, a switch to the next thread will be performed..." *Figure 3, elements 112 and 114* Parady has taught that the thread switching logic 112 switches threads when it detects a long-latency event indication on line 114. The long-latency event requires the pipeline to stall immediately to resolve the problem, thereby causing latency in the instruction execution. Parady switches threads when this latency occurs, so, instead of the processor being idle while the long-latency problem resolves, useful work is being performed by the processor. Parady uses the example that line 114 is an L2-miss indication, but this is to match the embodiment used as an example in Parady's Description of the Preferred embodiment. Parady states in column 2, lines 20-22 that "In one embodiment, the long-latency events are load or store operations..." but the device is for long-latency events in general (Parady column 2, lines 18-20). Therefore, Parady detects a long-latency event by determining if there is an indication of whether a long-latency on line 114 or not. Parady's thread switching logic 112 schedules the thread switch to happen immediately, i.e. appoints the thread switch to execute at that moment in time, when a switching event, i.e. long-latency event, occurs. **Bondi**: *Column 1, lines 47-64* Parady has not explicitly taught that the long-latency event is a mispredicted branch, so has not taught that the switch logic module detects a branch misprediction and schedules a switch

	to another thread during a latency of the branch misprediction. However, as seen in the above paragraph, Parady has taught that his invention detects a long-latency event and schedules a switch to another thread during a latency of the long-latency event. Parady has also taught that a jump instruction, which is the same as a branch instruction (InstantWeb's Free On-line Computing Dictionary "branch"), is a potential long-latency event (Parady column 4, lines 6-8). Also, Parady has taught that a thread switch occurs on a long latency event (see above) and multi-threading reduces the impact of long latency events (Parady column 1, lines 58-59). In the column 1, lines 47-64, Bondi has taught that branch misprediction requires "numerous cycles...to reset the pipeline(s) to an operational state and, thus, valuable processor cycle time is lost." In essence, Bondi has taught that a branch misprediction is a long latency event, since it requires numerous cycles to resolve, and it is "one of the more serious impediments to realizing even higher processor performance (Bondi column 1, lines 62-64)." A person of ordinary skill in the art would have recognized that a branch mispredict is a type of long latency event and reducing the performance penalty due to the misprediction would increase processor performance. Therefore, it would have been obvious to modify Parady to switch threads on a branch misprediction in order to reduce the performance penalty suffered by the misprediction.

Referring to claim 2:

Claim Limitation	Cited Prior Art and Explanations
Wherein the switch logic module detects a switching event	**Parady** *Column 3, lines 58-63* "The indication that a thread switch is required is provided on a line 114...Upon such an indication, a switch to the next thread will be performed..." *Figure 3, elements 112 and 114* Parady has taught that the thread switching logic 112 switches threads when it detects a long-latency event indication on line 114. Parady uses the example that line 114 is an L2-miss indication, but this is to match the embodiment used as an example in Parady's Description of the Preferred embodiment. Parady states in column 2, lines 20-22 that "In one embodiment, the long-latency events are load or store operations..." but the device is for long-latency events in general (Parady column 2, lines 18-20). Therefore, Parady

	detects a long-latency event by determining if there is an indication of whether a long-latency on line 114 or not.

Referring to claim 3:

Claim Limitation	Cited Prior Art and Explanations
Wherein the switch logic module includes an indicator that is set when a branch misprediction is detected and reset when the switch is completed	**Parady** *Column 2, lines 18-19* "The present invention provides a method and apparatus for switching between threads of a program in response to a long-latency event." *Column 2, lines 27-29* "A switching mechanism switches between the program address registers in response to the long-latency events." *Column 3, lines 58-63* "The indication that a thread switch is required is provided on a line 114…Upon such an indication, a switch to the next thread will be performed…" *Figure 3, elements 112 and 114* Parady teaches that there is an indication (Parady Figure 3, element 114) that shows whether a switch should occur in response to or not. The indication signals whether there is a long latency event (Parady column 3, lines 57-65). This indication is inherently set when a long latency event occurs to signal the thread switch logic 112 to switch threads. This indication is then inherently reset when the switch is complete. If the indication remained set after a thread switch, then the thread switch logic 112 would switch continuously in response to the indication. **Bondi**: *Column 1, lines 47-64* Parady has not explicitly taught that the long-latency event is a mispredicted branch, so has not taught that the switch logic module detects a branch misprediction and schedules a switch to another thread during a latency of the branch misprediction. However, as seen in the above paragraph, Parady has taught that his invention detects a long-latency event and schedules a switch to another thread during a latency of the long-latency event. Parady has also taught that a jump instruction, which is the same as a branch instruction (InstantWeb's Free On-line Computing Dictionary "branch"), is a potential long-latency event (Parady column 4, lines 6-8). Also, Parady has taught that a thread switch occurs on a long latency event (see above) and multi-threading reduces the impact of long latency events

	(Parady column 1, lines 58-59). In the column 1, lines 47-64, Bondi has taught that branch misprediction requires "numerous cycles...to reset the pipeline(s) to an operational state and, thus, valuable processor cycle time is lost." In essence, Bondi has taught that a branch misprediction is a long latency event, since it requires numerous cycles to resolve, and it is "one of the more serious impediments to realizing even higher processor performance (Bondi column 1, lines 62-64)." A person of ordinary skill in the art would have recognized that a branch mispredict is a type of long latency event and reducing the performance penalty due to the misprediction would increase processor performance. Therefore, it would have been obvious to modify Parady to switch threads on a branch misprediction in order to reduce the performance penalty suffered by the misprediction.

Referring to claim 4:

Wherein the switch logic module includes an outstanding switch request indicator that is set when the switching event does not require an immediate switch	**Parady** *Column 2, lines 45-59* "One techniques for limiting the effect of slow memory accesses is a 'non-blocking' load or store...means that other operations can continue in the processor while the memory access is being done. Other load or store operations are 'blocking'...meaning that processing of other operations is help up while waiting for the results of the memory access..." *Column 4, lines 53-62* "...the present invention also supports non-blocking loads which allow the program to continues in the same program thread while the memory access is being completed...would be supported in addition to blocking loads...there would not be a thread switch immediately on a non-blocking load, but would be upon becoming a blocking load waiting for data..." Examiner notes that Parady's description of non-blocking and blocking loads is similar to Applicant's description of non-blocking and blocking loads on page 5, lines 24-26 and on page 6, line 31 to page 7, line 6. By characterizing the load instructions into two categories and being able to distinguish between the two, Parady's system recognizes that non-blocking loads do not require an immediate thread switch and blocking loads require an immediate thread switch. Parady would inherently need some type of indicator that distinguishes a non-blocking load that does not require an

	immediate thread switch, i.e. does not schedule the thread switch to occur immediately, and a block load that does require an immediate thread switch, i.e. does schedule the thread switch to occur immediately.

The following shows what the examiner considers equivalent limitations found in claim 1, which is the device claim of the apparatus, and in claims 8 and 14, which are the method claims of the same apparatus. The rejection used for the limitation in claim 1 explained in detail in the above table and in the response to arguments below applies to the equivalent limitation in claims 8 and 14. Emphasis was added by the Examiner.

Claim 1	Claim 8	Claims 14
A multi-threading processor, comprising:	A method for concealing switch latency in a multi-threading processor, comprising:	A set of instructions residing in a storage medium, said set of instructions capable of being executed by a processor for concealing switch latency in a multi-threading processor comprising:
A front end module		
An execution module coupled to said front end module and said execution module; and		
A switch logic module coupled to said state module		
Wherein said switch logic module <u>detects a mispredicted branch in a software thread</u>	<u>Detecting a switching event in a software thread</u>; <u>Determining whether a mispredicted branch has been detected</u> in said software thread; and	<u>Detecting a switching event in a software thread</u>; <u>Determining whether a mispredicted branch has been detected</u> in said software thread; and
And schedules <u>a switch to another software thread during a latency of said mispredicted branch</u>	Executing <u>a switch to another software thread during a latency of said mispredicted branch</u> if said mispredicted branch has been detected.	Executing <u>a switch to another software thread during a latency of said mispredicted branch</u> if said mispredicted branch has been detected.

Please see the rejection and table corresponding to claim 1 for more information on the rejections of claims 8 and 14. In further detail, the switching event is a branch misprediction, since it requires numerous cycles to resolve. Parady has taught that his switching event is a long-latency event (Parady column 2, lines 18-20) and that a jump instruction, also known as a branch instruction (InstantWeb's Free On-line Computing Dictionary "branch"), is a possible long-latency event (Parady column 4, lines 6-8). Bondi has taught that a mispredicted branch is a long-latency event (Bondi column 1, lines 47-64). Also, a mispredicted branch is a type of branch that is a long-latency event. Therefore, the rejection to claim 1 is applicable to claims 8 and 14.

(10) Response to Argument

Prior to responding to each individual argument, the Examiner would like to make one general observation about the arguments made. In general, the arguments were in regard to details which were not claimed, but that the Applicant seems to believe were inherent to the definition of the claim language. The language relied upon were not defined in the specification in any form that allowed the Examiner to adopt the specific meaning alluded to in the arguments, so a general definition in the art was applied. For example, the definition Applicant seemed to allude to in his arguments of "state module" in claim 1 had a specific definition in the specification that stated the "state module" must be one unit with a PC and register as shown in Applicant's Figure 1. However, there is nothing in the claim language or specification that states the "state module" cannot be Parady's Figure 3, elements 110, 48, and 50 and must be something similar to the single "state module" shown in Applicant's Figure 1.

Applicant argues in essence on pages 3-4,

> The Examiner has admitted that neither the Parady reference nor the Bondi reference teaches or suggests the noted claim elements. Therefore, no possible combination of Parady and Bondi can render claim 1 obvious because the admittedly missing claim element is not taught or suggested by the combination (page 4, paragraph 1).

This has not been found persuasive. Please see the table for claim 1 above for details of the rejection. Parady has taught that switch logic module detects a long latency event in a software thread and schedules a switch to another software thread during a latency of said long latency event (Parady column 2, lines 18-19 and column 2, lines 27-29). Parady has even taught that jump instructions, which are equated to branch instructions (InstantWeb's Free On-line Computing Dictionary "branch"), are long latency instructions (Parady column 4, lines 6-8). Parady has not taught that "branch misprediction" is one of his set of long-latency events. Bondi has taught that branch misprediction requires "numerous cycles...to reset the pipeline(s) to an operational state and, thus, valuable processor cycle time is lost." In essence, Bondi has taught that a branch misprediction is a long latency event, since it requires numerous cycles to resolve, and it is "one of the more serious impediments to realizing even higher processor performance (Bondi column 1, lines 62-64)." Parady has taught that a thread switch occurs on a long latency event (see above) and multi-threading reduces the impact of long latency events (Parady column 1, lines 58-59). A person of ordinary skill in the art would have recognized that a branch mispredict is a type of long latency event and reducing the performance penalty due to the misprediction would increase processor performance. Therefore, it would have been obvious to modify Parady to switch threads on a branch misprediction in order to reduce the performance penalty suffered by the misprediction. The test of obviousness is not what has been taught explicitly by each reference, as suggested by Applicant's arguments, but what the two references

suggest to a person of ordinary skill in the art at the time the invention was made (In re Bozek, 163 USPQ 545 (CCPA 1969) "The test for obviousness is not whether the features of one reference may be bodily incorporated into the other to produce the claimed subject matter by simply what the combination of references makes obvious to one of ordinary skill in the pertinent art."; In re Van Beckum, 169 USPQ 47 (CCPA 1971) "We would note that it is well settled that the test of obviousness is not whether the features of one reference can be bodily incorporated in to the structure of another and proper inquiry should not be limited to the specific structure shown by the references, but should be into the concepts fairly contained therein, and the overriding question to be determined is whether those concepts would suggest to one skilled in the art the modifications called for by the claims."; In re Sheckler, 168 USPQ 716 (CCPA 1971) "...It is, of course, not necessary that either Barnes or Dryden actually suggest, expressly or in so many words, the changes or possible improvements appellant has made").

Applicant argues in essence on pages 4-6

> ...While 'comprising' is open claim language, it does not operate to negate the recited inter-relationship of the claim elements. Particularly from the view of the claim as whole, the structure and inter-relationships of the components in Parady are different from and do not teach or suggest the recited modules and inter-relationship of the modules as recited in claim 1 (page 6, paragraph 2).

This has not been found persuasive. Claim 1 has three main modules that have an "inter-relationship":

1. an execution module coupled to said front end module
2. a state module coupled to said front end module and said execution module; and
3. a switch logic module coupled to said state module

Referring to Parady's Figure 3, the execution module is the Execution Units 41. The state module is the PA Registers 110, Integer Register File 48, and FP Register Files 50. The front end module is the decode unit 14. The switch logic module is thread switching logic 112. Figure 3 shows a connection between the decode unit 14 and the execution units 41 via the instruction buffers 102-108, dispatch unit 28, and the busses/connections between these elements, thereby showing a coupling between the front end module and execution module. There is also a connection between the PA Register Files 110 and the decode unit 14 via the instruction buffers 102-108 and a connection from the Integer Register Files 48 and FP Register Files 50 to the Execution Units 41, thereby coupling the state module to the front end module and execution module. Finally, there is a connection between the thread switching logic 112 and the register files 110, 48, and 50, thereby coupling the switch logic and state modules. As can be seen by the explanation above and Parady's Figure 3, the "inter-relationship" of the modules exists in Parady. Also, in regards to the argument "...the Examiner relies on an unreasonable broad interpretation of 'comprising' to render any structural recitation" on page 6, the term "comprising" is open-ended and allows for additional elements to be present in the case, according to MPEP 2111.03. There is no limitation to the amount of extra elements that may be present in an open-ended claim (*Ex parte Davis*, 80 USPQ 448, 450 (Bd. App. 1948)). "Coupled" in the language of the claims does not require a direct connection between two elements, but only some type of connection so data can travel from point A to point B. There is no limitation in the word "coupled" as to the number of elements the data can travel through before reaching its destination as long as the data reaches its destination. The Applicant seems to be reading the language "coupled" to a much narrower definition then it is normally warranted

without any support in its claims or specification (<u>In re Self</u>, 671 F.2d 1344, 213 USPQ 1, f (CCPA 1982) "...Thus, it is immaterial that Binckley is a variable head device, while the claimed invention is contemplated to be a constant head device, because appellant's claims are not limited to the latter...It matters not that Binckley does not operate in the same way to accomplish the same result where appellant has not limited his claims according to function or result.")

Applicant argues in essence on pages 6-7 and 8

> ...Parady fails to teach or suggest even a switch logic module <u>coupled to</u> the state module, wherein the switch logic module <u>detects</u> an event in a software thread and <u>schedules</u> a switch to another software thread <u>during a latency</u> of the event (page 7, paragraph 3)...

This has not been found persuasive. Please see the table for claims 1 and 2 above for details on the rejections. According to the specification, Applicant's invention in summary detects a long-latency event, which normally causes a thread switch, but schedules the switch to occur during the latency of a branch misprediction. This means that when the long-latency event detected is a branch misprediction, as in the claims ("wherein said switch logic module detects a mispredicted branch in a software thread..."), the switch must happen immediately, since the switch occurs during the latency of the branch misprediction. As shown in the above response and in Parady Figure 3, the switch logic module is coupled to the state module. The switch logic module does detect an event in a software thread. Specifically, the switch logic detects when the long latency event indication for a software thread is true (Parady column 3, lines 57-65). The indication sent to the switch logic on line 114 in Parady's Figure 3 is detected by the switch logic and the switch logic performs the thread switch immediately, i.e. schedules the thread switch for immediate execution. To detect means "to discover or ascertain the existence, presence, or fact

of" and to schedule means "to plan or appoint for a certain time" (American Heritage Dictionary ©2000). The Applicant seems to be suggesting that there is a narrower definition of "detect" and "schedule" than in the claim, but there is nothing more specific in the claim language or in the specification to give the terms "detect" and "schedule" this narrower interpretation. Parady's switch logic 112 detects whether an indication that a long-latency event is present via line 114 and schedules a thread switch for immediate occurrence. Parady's Figure 3, element 114 is labeled L2-miss signal, but Parady in column 2, lines 18-22 states that his invention is for any long-latency event and that the long-latency event of an L2 cache miss is just one embodiment or example. The L2-miss signal is really just the signal which indicates when a long-latency event has occurred allowing a receiver of that signal to "detect" (ascertain the existence) of a long-latency event.

Applicant argues in essence on page 8 "Parady does not even mention and is not concerned with the problem of latency of mispredicted branches." This has not been found persuasive. Parady and Bondi both deal, in general, with the problems of latency. As shown in Bondi in the pages cited above, branch misprediction is merely one kind of latency problem. Parady teaches how to deal with latency problems, Bondi teaches one kind of latency problem, and Applicant's invention also teaches how to deal with latency problems.

Applicant argues in essence on pages 7-8

> ...the combination still fails to establish a prima facie case of obviousness because even assuming, for the sake of argument, that Bondi provides motivation to modify Parady, Bondi fails to teach or suggest how to modify Parady in a manner that might read on the claims (page 8, paragraph 1).

This has not been found persuasive. Please see the above table for claim 1 for details about the rejection. A combination does not mean to literally put one invention in another

invention, but, rather, what the two inventions together would suggest to one of ordinary skill in the art. The Parady invention teaches multi-threading and switching threads on a long latency event to improve processor performance. Bondi teaches that branch mispredictions are long latency events that degrade the performance of a processor. A person of ordinary skill in the art at the time the invention was made would have recognized that the performance penalty of branch mispredictions is "one of the more serious impediments to realizing even higher processor performance (Bondi column 1, lines 62-64)" and that Parady improves processor performance by trying to reduce the performance penalty of long latency events (Parady column 1, lines 58-59). Therefore, a person of ordinary skill in the art would have recognized that the long latency events of Parady includes the branch misprediction of Bondi and would have modified Parady include mispredicted branches in order to minimize the branch misprediction penalty that Bondi has taught impedes higher processor performance. How to modify a primary reference does not necessarily need to come from the secondary reference. It is what the two references, when held together by a person of ordinary skill in the art, would have suggested to the person of ordinary skill in the art (In re Bozek, 163 USPQ 545 (CCPA 1969) "The test for obviousness is not whether the features of one reference may be bodily incorporated into the other to produce the claimed subject matter by simply what the combination of references makes obvious to one of ordinary skill in the pertinent art."; In re Van Beckum, 169 USPQ 47 (CCPA 1971) "We would note that it is well settled that the test of obviousness is not whether the features of one reference can be bodily incorporated in to the structure of another and proper inquiry should not be limited to the specific structure shown by the references, but should be into the concepts fairly contained therein, and the overriding question to be determined is whether

those concepts would suggest to one skilled in the art the modifications called for by the claims."; In re Sheckler, 168 USPQ 716 (CCPA 1971) "...It is, of course, not necessary that either Barnes or Dryden actually suggest, expressly or in so many words, the changes or possible improvements appellant has made").

Claim 3

Applicants argue in essence on page 9 "...the office action is incorrect in its assertion...with respect to resetting the L2-miss signal when the switch is completed (Page 9, paragraph 1)." This has not been found persuasive. Please see the above table for claim 3 for details about the rejection. Parady teaches that there is an indication (Parady Figure 3, element 114) that a thread switch is to occur due to a long latency event (Parady column 3, lines 57-65). This indication is inherently set when a long latency event occurs to signal the thread switch logic 112 to switch threads. This indication is then inherently reset when the switch is complete. If the indication remained set after a thread switch, then the thread switch logic 112 would switch continuously since it would have constantly detected an indication to switch.

Claim 4

Applicant argues in essence on page 9 "...the recited outstanding switch request indicator is not inherent in Parady. Moreover, as noted above, the logic 112 does not schedule the switches, it merely performs them (page 9, paragraph 2)." This has not been found persuasive. Please see the table for claim 4 above for details about the rejection. Parady has categorized load instructions into two categories: non-blocking and blocking. Blocking loads are instructions which require immediate thread switches, since they are a long latency event that triggers a thread switch. Non-blocking loads are loads which do not require an immediate thread switch,

but could require a thread switch in the future if it becomes a blocking load (Parady column 1, lines 45-59). Parady has taught in column 4, lines 53-62 that his system would distinguish between the two types of load operations, so there must be some type of indicator that the load is a blocking or non-blocking load. This means that there must be some indicator inherent to the load instructions which signal whether it is blocking or non-blocking, i.e. requiring an immediate thread switch or a possible future thread switch. An outstanding switch request indicator is merely a switch request indicator that has not been resolved. The non-blocking loads have a switch indicator that is not known to be needed or not needed. As can be seen in Parady Figure 4 and in column 3, line 66 to column 4, line 6, the load instructions come with two bits that indicate the next thread pointer. A non-blocking load does not know whether it will need the indicator to the next thread unless it becomes a blocking load. Please refer to the above argument with regards to whether the logic 112 schedules a switch or not.

Claims 8 and 14

Applicant argues in essence on pages 9-10

> ...the Examiner has failed to identify what portion(s) of the references are relied upon for teaching detecting a switching event in a software thread and determining whether a mispredicted branch has been detected in said software thread (page 9, paragraph 2).

This has not been found persuasive. Please see the tables for claims 1, 8, and 14 above for the details of the rejection. The table below shows how the limitations of claim 1 map to the limitations in claim 8 and 14. The Examiner added the emphasis.

Claim 1	Claim 8	Claims 14
A multi-threading processor, comprising:	A method for concealing switch latency in a multi-threading processor, comprising:	A set of instructions residing in a storage medium, said set of instructions capable of being executed by a processor

		for concealing switch latency in a multi-threading processor comprising:
A front end module		
An execution module coupled to said front end module and said execution module; and		
A switch logic module coupled to said state module		
Wherein said switch logic module detects a mispredicted branch in a software thread	Detecting a switching event in a software thread; Determining whether a mispredicted branch has been detected in said software thread; and	Detecting a switching event in a software thread; Determining whether a mispredicted branch has been detected in said software thread; and
And schedules a switch to another software thread during a latency of said mispredicted branch	Executing a switch to another software thread during a latency of said mispredicted branch if said mispredicted branch has been detected.	Executing a switch to another software thread during a latency of said mispredicted branch if said mispredicted branch has been detected.

As can be seen in the table, the limitations of claims 8 and 14 are similar in scope to the limitations of claim 1 regarding the switch logic module. Claims 8 and 14 are broader in scope, since they do not contain the limitations regarding the front end, execution, and switch logic modules' couplings. Therefore, the rejection used on claim 1 is applicable as the rejections to claims 8 and 14, since the limitations found in claims 8 and 14 were already rejected in claim 1. The same art and citations from the prior art are applicable for claims 8 and 14. As a side note, the Examiner wishes to note that Parady's invention runs programs in an OS, which means that Parady's threads are software threads (Parady column 4, lines 31-41). Also, the Examiner wishes to point out that it does not matter whether the thread is a software thread or a hardware thread, since they function the same and what is executed in software can be done in hardware and what is done in hardware can be done in software. Please see Tanenbaum's Structured Computer Organization for more information.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

Aimee J. Li

April 18, 2005

Conferees
Eddie Chan

EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

Donald Sparks

DONALD SPARKS
SUPERVISORY PATENT EXAMINER

KENYON & KENYON
Suite 600
333 W. San Carlos Street
San Jose, CA 95110-2711